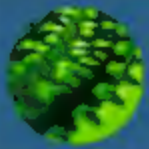


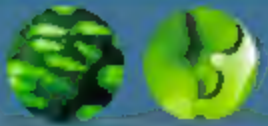
Singularity

Galen Hunt
James Larus
David Tarditi
Microsoft Research

MSR Cool Talk
September 15, 2005

"Modern" OS & Applications

- Architecture, languages, tools designed in 1960's & 1970's
- Design parameters
  - resources scarce
  - environment benign
  - users knowledgeable and well trained



The World Changed

- Machines are fast, memory is cheap
- Ubiquitous computing means ubiquitous worms, viruses, scams, attacks, …
- Few users understand computers or software

Presentation

- Overview (Jim)
- Singularity OS (Galen)
- Language, compiler & runtime (David)
- Opportunities (Jim)

Singularity

- New OS, programming language, tools from MSR
  - goal: more dependable system
  - attack problem from multiple directions
- Key points
  - advances in languages, compilers, and tools open the possibility of improving software
  - Singularity uses these advances to build more dependable systems and applications
  - systems built on Singularity expand software delivery opportunities
  - Research vehicle, not Windows replacement!

Technological Advances

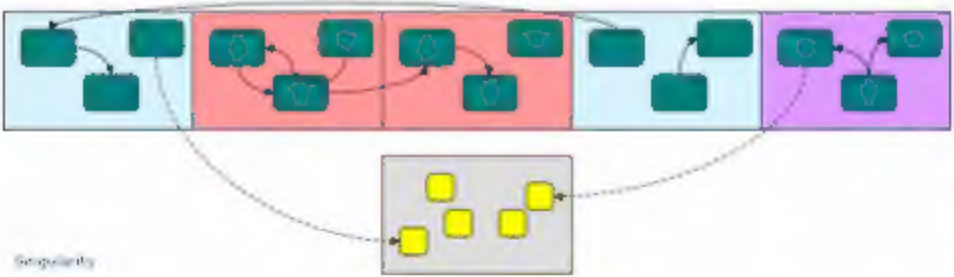
- Integral support for correctness and verification
  - safe programming languages
  - optimizing compilers to reduce safety penalty
  - end-to-end validation of program properties
  - sound, specification-driven correctness tools

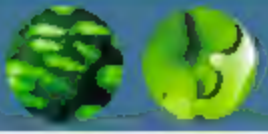
New System Architecture

- Major features
  - lightweight isolation and fault containment
  - channels for communication and interaction
  - integration of managed code runtime and OS
  - self-describing system
  - programming language extensions to improve dependability

Software Isolated Processes (SIPs)

- Lightweight managed code environment (not CLR)
  - only executes safe, managed code
  - fast process creation and communications
  - protection and failure isolation domain
- Closed object (not address) space
  - enforced by language safety and interface design
  - global invariant: no process contains a pointer to another object space
- Rely on language safety (not VM hardware)
  - system written in C# dialect (safe outside of kernel)





Verification

- Detect errors early, trust late
- Verification serves two roles
  - prevent and detect programming errors
  - ensure system integrity

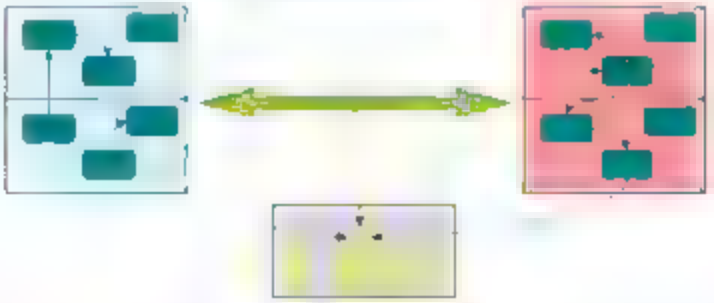
Channels

- Verifiable, efficient communications
  - channel contract specifies value and communication protocol
  - verify code obeys contract (compile time)
  - check communications against contract (runtime)
- Kernel controls establishment of channels
- Security policy tied to channels

Channel Communication

- Key is "send & forget" message semantics
  - message owned by at most one process
    - ownership transferred at send (no copies)

    facilitates static protocol verification
  - enables efficient implementation



Process Independence

- SIPs on Singularity are fully independent
  - no shared implementation or state (outside kernel)
    - no shared memory
    - garbage collector and runtime chosen to fit application
  - GCs run independently
  - OS terminates and reclaims resources
- High degree of isolation
  - resource usage independently mediated by OS
    - e.g., can't force GC in another process
    - kernel API does not allow one process to affect another process
  - interact only through visible, system-mediated channels

Failure Isolation

- Process is failure boundary
  - no shared data structures
  - clean failure notification on channel
  - resources reclaimed by OS
- Communication partners should recover and continue
  - recovery is feasible, not transparent
  - e.g. device driver failure causes glitch

Sealed Minimal Environment

- No dynamic code loading or run time code generation
  - all code present when process starts
    - enable sound program analysis, optimization, compilation
    - extensions run in separate process
  - compile-time reflection (CTR) is part a replacement for code generation
- Environment starts empty
  - SIP without channel cannot affect anything else
  - limit libraries to those appropriate for application
    - enforce design discipline
    - enforce system and security policy

Uniform Extension Mechanism

- SIPs used throughout system and applications
  - device drivers, system services, applications, extensions, …
- Single, general mechanism
  - implement correctly and efficiently
  - build language and tools support

Security Model

- Security at process granularity (not CAS)
- Principal is triple
  - machine
  - user
  - application (process)
- ACLs on resources
- Access checks when communication established
  - e.g. channel to read/write file
  - can determine party at other end of channel
  - delegate permission by passing established channel

Applications

- Application are first-class OS abstraction
  - code + resources + manifest
  - system controls installation
    - verify code & manifest
    - check for conflicts
- Manifests describe components and dependencies

Singularity Project

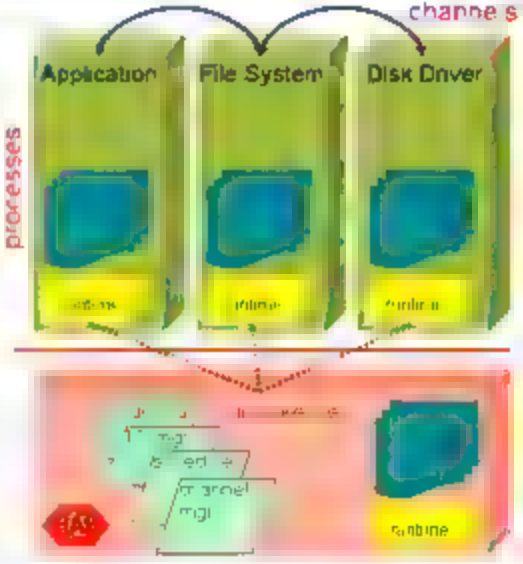
- Develop technology and infrastructure to build more dependable software
  - language support to increase software quality
    tools to ensure correct software behavior
    OS architecture that enhances system dependability
  - superior dependability, sufficient performance
- Large research project
    ~dozen researchers & RSDEs
    Redmond, Silicon Valley, Cambridge
- System running on hardware and Virtual PC
- Not Windows successor!

Presentation

- Overview (Jim)
- Singularity OS (Galen)
- Language, compiler & runtime (David)
- Opportunities (Jim)

Maximizing Safe Code

- Dependable Kernel
  - kernel closed at boot time
  - OS services moved to processes
    device drivers moved to processes
    -
  - reduced use of unsafe code
    -
    -
    -
- Dependable processes
  - processes closed at start time
    only trusted runtime code can use unsafe C# features
  - application code is 100% safe
    service code is 100% safe
    device driver code is 100% safe



Controlling Access to Hardware

- Hardware resources are accessed through classes in the trusted runtime

  `IoPort`, `IoIrq`, `IoMemory` and `IoDma`

  `IoPortRange`, `IoIrqRange`, `IoMemoryRange`, and `IoDmaRange`

- The trusted runtime permits the creation of I/O objects to resources allocated by kernel to driver through an activation object

- Methods on I/O objects verify access before operating directly on hardware

- For example:

```
IoPortRange myDeviceRange = IoConfig.GetConfig().DynamicRange[0];
IoPort maskPort = myDeviceRange.PortAtOffset(4, 1, Access.Write);
maskPort.Write8(0xff);
```

- Open research: How do we describe and verify register usage at hardware, software interface?

Enabling Verification of Application Dependencies

- Applications declare their system requirements and products using a set of custom attributes and an XML schema
  - code assemblies required
    channels required
    channels produced
  - hardware resources required (if driver)
  - hardware resources enumerated (if bus driver)
- The OS uses this application metadata to
  - verify that an application is valid on this system
    detect and resolve conflicts based on declared system policies
    bind global resources to private names in activation objects

Application Attributes

PCI Device

Dynamic resources from PCI config (frame buffer)

```
[DriverCategory]
[Signature("/pci/03/00/5333/8811")]
class S3Trio64Resources : DriverCategoryDescription
{
  [IoMemoryRange Base = 0x... Length = 0x400000 ]
  IoMemoryRange frameBuffer,

  [IoFixedMemoryRange Base = 0xB8000 Length = 0x8000 ]
  IoMemoryRange textBuffer,

  [IoFixedPortRange Base = 0x3B0 Length = 0x2C ]
  IoPortRange control,

  [ExtensionEndpoint(typeof(ExtensionContract.Exp ))]
  TRef<ExtensionContract.Exp.Start> pnp

  [ServiceEndpoint(typeof(VideoDeviceContract.Exp ))]
  TRef<ServiceProviderContract.Exp.Start> video,
  ...
```

Fixed resource (VGA buffer)

Fixed resource (SVGA I/O Ports)

Requires channel to parent process for control

Produces channel for clients to access video device

Application Manifest

PCI device

List of MSIL assemblies

Channels

Fixed resources

Dynamic resources

```
<driver identity= S3Trio64  signature= /pci/03/00/5333/8811>
 <assemblies>
  <assembly filename="S3Trio64.exe"      />
  <assembly filename= Namespace Contracts.dll      />
  <assembly filename= Io Contracts.dll      />
  <assembly filename="Corlib.dll      />
  <assembly filename="Singularity.V1.dll      />
 </assemblies>
 <endpoints count= 2">
  <extension startStateId= 3 endpointEnd= Exp
   contractName= Microsoft.Singularity.Extending.ExtensionContract
   assembly= Namespace Contracts  index= 0  />
 </endpoints>
 <fixedResources count= 3 >
  <ioMemoryRange base= 0xb8000  length="0x8000" index="0" />
  <ioPortRange base= 0x3c0  length= 0x20  index= 2   >
 </fixedResources>
 <dynamicResources count="1 >
  <ioMemoryRange default= 0xf8000000  length= 0x400000  index="0   >
 </dynamicResources>
```

Metadata-Driven Activation

Execution in kernel

```
// I/O manager creates and fills activation record
IoConfig config = new IoConfig();
config.DynamicRanges[0] = new IoMemoryRange(0xd0000000, 0xB00000); // PCI
config.FixedRanges[0] = new IoMemoryRange(0xb8000, 0x8000);
config.FixedRanges[2] = new IoPortRange(0x3c0, 0x20);
process.IoConfig = config;

process.AddEndpoint(NameSpace.Bind("/dev/video"));
```

Trusted execution in driver process

```
// activation object filled from activation record
frameBuffer = (IoMemoryRange)IoConfig.GetConfig(DynamicRanges[0]);
textBuffer = (IoMemoryRange)config.GetConfig(FixedRanges[0]);
control = (IoPortRange)config.GetConfig(FixedRanges[2]);

video = new TRef<ServiceProviderContract.Exp>(Starts.Exp.GetEndpoint());
```

```
ServiceProviderContract.Exp video = S3Trio64Resources.video.Acquire();
video.RecvConnect(out client);
```

Application code in driver

Verifying Kernel Dependencies

- Without "formal" API description
  - app compat bugs abound
  - app dependencies aren't reliable
- Application Binary Interface (ABI) is
  - minimal interface (~150 functions)
  - defined by compiled IL* assembly inscribed into app at compile time
- ABI is a functional (not OO) API
  - only value types cross ABI
  - separates app and kernel GC domains
  - wrapped in BCL for ease of use
- ABI maintains process independence
  - cross process operations exclusively through channels
- ABI is layer of indirection for replacing implementation

But is it efficient enough to use?

YES!

	Cost (CPU Cycles)			
	Singularity	FreeBSD 5.3	Linux 2.6.9	Windows XP
Minimum kernel API call	201	861	547	443
Message request/reply	2,379	15,815	23,618	(LPC) 4,653 (IPC) 13,371
Process create & start	218,236	807,581	658,911	7,231,038

- Why?
  - Because all SIPs run in ring 0
  - Because static verification replaces hardware protection
  - Because we use a good optimizing compiler (not JIT)

Presentation

- Overview (Jim)
- Singularity OS (Galen)
- Language, compiler & runtime (David)
- Opportunities (Jim)

Language, Compiler and Runtime

- Sing# is extended dialect of C#
  - built on Spec#
    - extensions for systems programming, program specification, and verification
- Bartok Research Compiler
  - highly optimizing ahead-of-time compiler compiles MSIL to native x86 machine code
- Bartok Runtime System
  - lightweight, customizable run-time system
- Verifier
  - extended version of MSIL verification

# of Language Features

Supported

- Core types + object model instructions
  - integers, floats, objects, managed pointers, unmanaged pointers
- Garbage collection including finalizers, weak references
- Exception handling
- Lazy type initialization
- Delegates
- Unsafe code (kernel/runtime use only)
- Data layout attributes
- Generics (coming soon)

Not Supported

- Dynamic class loading
- Reflection (very limited support)
- Code access security
- Platform invoke

Proposed Extensions to System.Runtime

- Control over class initialization
  - require a class constructor be run at process start up
  - specify initialization order
- [NoHeapAllocation] attribute
  - Compile time check that a method (and its callees) does not heap allocate
- Overlays for type safe structural casts of arrays of scalars
- Struct inheritance
  - allow a struct to inherit all the fields/methods of a parent struct
- And more

Language Support for Message Passing

- Extending languages simplifies coding and facilitates static checking
- Channel contracts specify valid protocol sequences and types for messages
- Switch-receive statement for asynchronous event pattern matching
- Exchangeable types for data passed by messages
  - data moves outside per-process GC heaps,

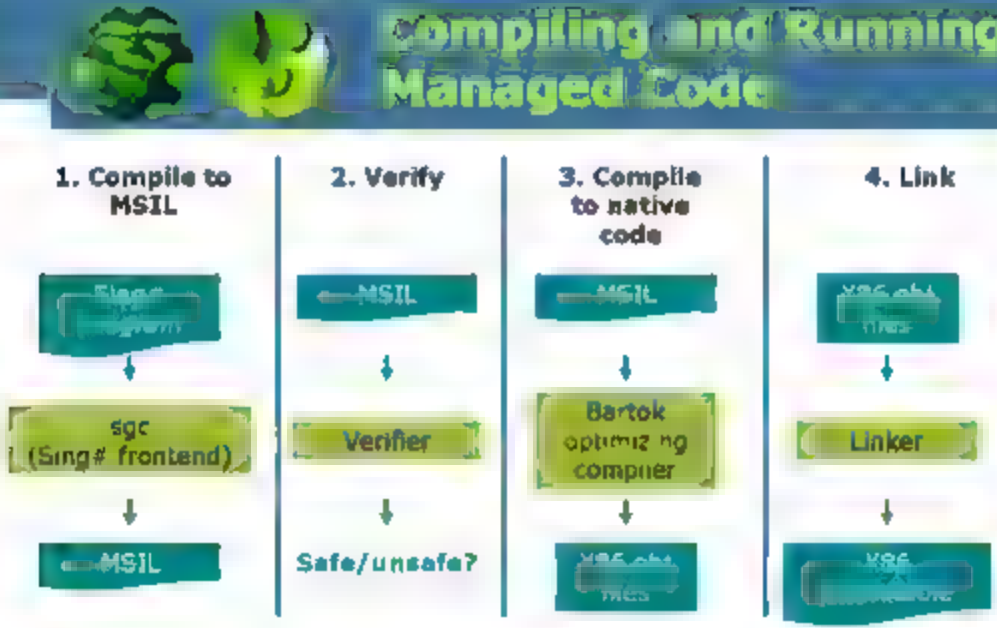
    ownership changes when sent across message
  - explicit resource management for exchangeable data and channel endpoints

    compiler verified, so it is safe

Channel contracts

- Contract declares
  - message types (name, argument types, direction)
  - state machine defining all valid message sequences
- Provides
  - channel endpoint types Imp, Exp
  - channel construction method
  - typed send and receive methods
- Efficient
  - pre allocated buffers
  - state machine and finite outstanding messages

Compiling and Running Managed Code



Compiling to Native Code

- Extensive optimization
  - 30+ optimizations: high-level, medium-level, code generation, runtime focused
- Assist specialization of runtime system
  - automatically remove unused or disabled language features

    tree shaking eliminates unused classes/methods/fields
- Whole program and separate compilation

  closed processes allow whole program optimization

  predecessor compiler (Marmot) demonstrated whole program performance competitive with C/C++

Managed Code Optimizations

- Tree shaking (whole program mode)
  - Eliminate unreachable or unused classes, interfaces, methods, and fields
- Array-bounds check elimination
- Array store check elimination
- Null check elimination
- Devirtualization of virtual calls
- Type-test and type cast elimination
- Optimize class initialization
  - Eliminate redundant checks
  - Fast slow path split
- Redundant field load store elimination
- Optimize convert operations
  - Widening narrowing introduced by use of argument stack
- Eliminate unused formal parameters
- Compress paths of struct operations

Lightweight Runtime System

1. Mark sweep
2. Mark sweep compact
3. Concurrent
4. Semi-space copying
5. Gen. copying
6. Reference counting

Automatic storage management

Threading and synchronization

Type tests

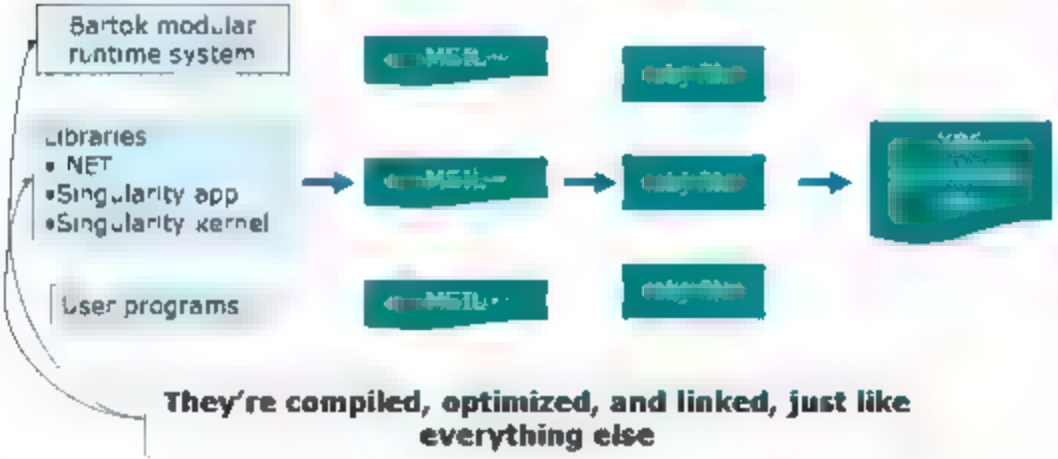
Exception handling

Interface calls

Virtual table and object layout

Core datatypes. integers, floating point, arrays, strings

Compiling Runtime System and Libraries



They're compiled, optimized, and linked, just like everything else

Dynamic Memory Usage: Hello World

	virtual address size (bytes)			
	Singularity	FreeBSD	Linux	Windows
C w/ static lib		232K	1,788K	663K
C++ w/ static lib		700K	2,372K	704K
C w/ GC	316K			3,750K

Presentation

- Overview (Jim)
- Singularity OS (Galen)
- Language, compiler & runtime (David)
- Opportunities (Jim)

Software as a Service

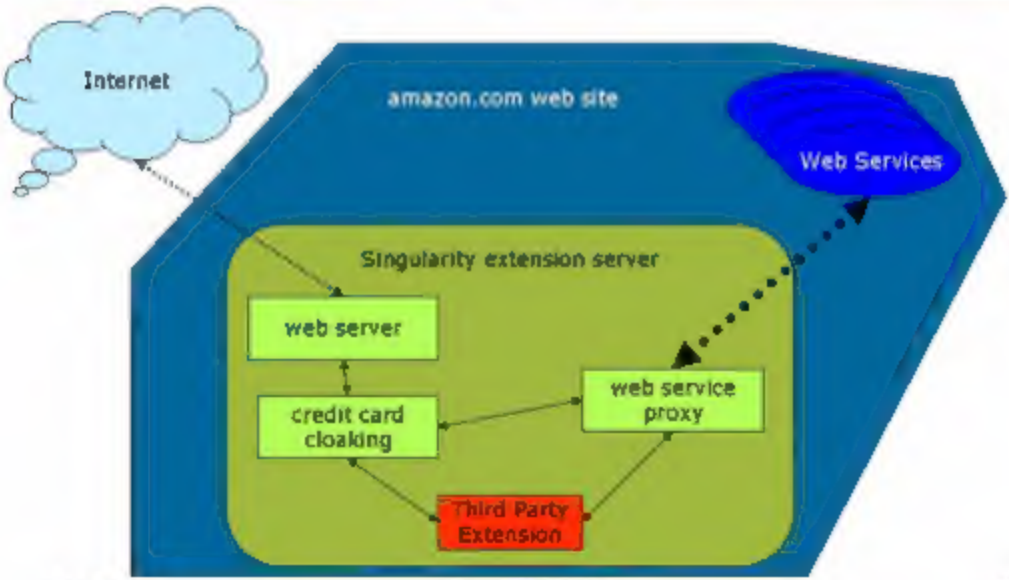
- Free users from system administration
- Deliver and remotely support software
  - $29.99/yr to support Office on your home machines?
- Why Singularity?
  - strong isolation
    - processes cannot interfere with each other
    - system has explicit and total control over communication mechanism
  - safe execution environment
    - OS controls verification, compilation and run-time libraries
  - remotely administrable
    - OS controls installation and configuration
    - detect conflicts before execution
  - security model
    - identify and protect application resources (files, metadata)

Web Extensions

- Developers cannot anticipate or satisfy all uses of their software
  - non-trivial software provides rich extension mechanisms (COM, VBA, ..)
- No safe way to host extensions to web site
  - e.g. Amazon.com storefronts or eBay bidding agent
- Developer with new idea must operate web site
- Why Singularity?
  - strong isolation
    - ...
  - safe execution environment
    - ...
  - remotely administrable
    - ...
  - security model
    - ...

Example

Safe Extensions to Web Sites



Questions?

- http://singularity
- DLs:
  - Singularity Design Notices: **singnote**
  - Singularity Questions and Answers: **singqa**
- Source access:
  - business need
  - or, willingness to contribute to the project
    - e.g., OSS model with Singularity team as filters